



TITLE:

Syntax (算法言語の設計-記述-処理の研究 : ALGOL N)

AUTHOR(S):

IGARASHI, S.; IWAMURA, T.; SAKUMA, K.; SHIMAUTI, T.; SHIMIZU, T.; TAKASU, S.; WADA, E.; YONEDA, N.

CITATION:

IGARASHI, S. ...[et al]. Syntax (算法言語の設計-記述-処理の研究 : ALGOL N). 数理解析研究所講究録 1969, 66: 9-18

ISSUE DATE:

1969-02

URL:

<http://hdl.handle.net/2433/107891>

RIGHT:

§1. Syntax

1.1 Meta-Language for Syntax Description

To describe the syntax of ALGOL N we use the following meta-language, which is a simple modification of the Backus notation.

1.1.1 Meta-symbols: $=, [,], \{, \}, |, /, \dots$

1.1.2 Meta-constants: begin, $;$, $($, a , etc.

Those are basic symbols of ALGOL N.

1.1.3 Meta-variables: $\langle \text{expression} \rangle, \langle \text{variable} \rangle, \langle \text{procedure call} \rangle$ etc.

Those are used to represent the form of syntactical elements.

1.1.4 Meta-expressions:

A meta-expression represents the forms of figures which are finite sequences of basic symbols, and are defined recursively as follows:

1) Let α stand for a meta-constant. Then

α

is a meta-expression.

A figure ϕ is of the form α , if and only if ϕ consists of just one basic symbol α .

2) Let α stand for a meta-variable. Then

α

is a meta-expression.

A figure ϕ is of the form α , if and only if ϕ is of the form meta-variable α .

3) Let α stand for a meta-expression. Then

(α)

is a meta-expression, and is used to express the precedence of connections.

A figure ϕ is of the form (α) , if and only if ϕ is of the form α .

- 4) Let α stand for a meta-expression. Then

$$\boxed{\alpha}$$

is a meta-expression.

A figure ϕ is of the form $\boxed{\alpha}$, if and only if ϕ is the empty figure or is of the form α .

- 5) Let α, β stand for meta-expressions. Then

$$\alpha\beta$$

is a meta-expression.

A figure ϕ is of the form $\alpha\beta$, if and only if ϕ is the concatenation of a figure of the form α and a figure of the form β .

- 6) Let α, β stand for meta-expressions. Then

$$\alpha|\beta$$

is a meta-expression.

A figure ϕ is of the form $\alpha|\beta$, if and only if either ϕ is of the form α or ϕ is of the form β .

- 7) Let α stand for a meta-expression. Then

$$\alpha \dots$$

is a meta-expression.

A figure ϕ is of the form $\alpha \dots$, if and only if either ϕ is of the form α or ϕ is the concatenation of a figure of the form $\alpha \dots$ and a figure of the form α .

- 8) Let α, β stand for meta-expressions. Then

$$\alpha\{\beta\} \dots$$

is a meta-expression.

A figure ϕ is of the form $\alpha\{\beta\} \dots$, if and only if either ϕ is of the form α or ϕ is the concatenation of a figure of the form $\alpha\{\beta\} \dots$, a figure of the form β , and a figure of the form α .

- 9) Let α, β stand for meta-expressions. Then

α/β

is a meta-expression.

A figure ϕ is of the form α/β , if and only if ϕ is of the form $\alpha|\beta|(\alpha\beta)$.

The ranking of the priorities of connections is as follows:

first: $\alpha \dots$, $\alpha\{\beta\} \dots$

second: $\alpha\beta$

third: $\alpha|\beta$, α/β

1.1.5 Meta-statement:

A meta-statement is used to define a meta-variable. Let α stand for a meta-variable, and β stand for a meta-expression. Then

$$\alpha \equiv \beta$$

is a meta-statement. This meta-statement represents the sentence:

"A figure ϕ is of the form of α , if and only if ϕ is of the form β ."

In the following, we shall say simply

" ϕ is α " in stead of " ϕ is of the form α ".

1.2 Standard Language of ALGOL N

<expression>	= <secondary> <form call>
<secondary>	= <primary> <array element> <structure element> <procedure call>
<primary>	= <variable> <go to statement> <dummy statement> <code call> <closed expression> <block> <notation>
<notation>	= <effect notation> <real notation> <bits notation> <string notation> <reference notation> <array notation> <structure notation> <procedure notation>
<declaration>	= <variable declaration> <form declaration> <mark declaration>

<go to statement>	= <u>go to</u> <label>
<dummy statement>	= <u>dummy</u>
<code call>	= <u>code</u> ([(<selector> <expression>) { , } ...] <primary typifier> <u>by</u> (<code body>))
<closed expression>	= (<expression>)
<block>	= <u>begin</u> [<declaration> ;] ... ([<label> :] ... <expression>) { ; } ... <u>end</u>
<array element>	= <secondary> [<expression>]
<structure element>	= <secondary> [<selector>]
<procedure call>	= <secondary> ([<expression> { , } ...])
<form call>	= [<expression>] <mark> [<expression>] { <mark> } ...
<effect notation>	= <u>effect</u>
<real notation>	= <u>real</u> <real modifier> <real donor>
<real modifier>	= [[<expression>] : [<expression>] : [<expression>]] [<u>precision</u> [<expression>]]]
<real donor>	= [<number>]
<number>	= <integer donor> <fraction donor>
<integer donor>	= <digit> ...
<fraction donor>	= [<digit> ...] (. <digit> ... / (₁₀ ⁺ ₁₀ ⁻) <digit> ...)

<bits notation>	= <u>bits</u> <bits modifier> <bits donor>
<bits modifier>	= [<u>1</u> (<u>exact</u> <u>varying</u>) [<expression>] <u>1</u>]
<bits donor>	= [<bits>]
<bits>	= (<u>0</u> <u>1</u>) ...
<string notation>	= <u>string</u> <string modifier> <string donor>
<string modifier>	= [<u>1</u> (<u>exact</u> <u>varying</u>) [<expression>] <u>1</u>]
<string donor>	= [<string>]
<string>	= ' [<non ' ' > <string>] ... '
<reference notation>	= <u>reference</u> <reference donor>
<reference donor>	= [<u>nil</u>]
<array notation>	= <u>array</u> <array modifier> <primary> array [<expression> { , } ...]
<array modifier>	= [<u>1</u> <expression> : <expression> <u>1</u>]
<structure notation>	= <u>structure</u> [[(<selector> <expression>) { , } ...] <u>1</u>]
<procedure notation>	= <u>procedure</u> ([<typifier> { , } ...]) <primary typifier> <procedure donor>
<procedure donor>	= [<u>by</u> (([<variable> { , } ...]) <expression>)]
<variable declaration>	= <u>let</u> <variable> <u>be</u> <expression>
<form declaration>	= <u>let</u> <form> <u>represent</u> <expression>
<form>	= [(<typifier>)] <mark> [(<typifier>)] { <mark> } ...

<mark declaration>	= <u>let</u> <mark> <u>operate</u> <left priority> <right priority>
<left priority>	= [<u>before</u> ([<mark> {,}...]) <u>all</u>] <u>left</u>
<right priority>	= [<u>after</u> ([<mark> {,}...]) <u>all</u>] <u>right</u>
<typifier>	= <expression>
<primary typifier>	= <primary>
<variable>	= <identifier>
<label>	= <identifier>
<selector>	= <identifier> :
<identifier>	= <letter> [<letter> <digit>]...
<letter>	= a b c d e f g h i j k l m n o p q r s t u v w x y z
<digit>	= 0 1 2 3 4 5 6 7 8 9
<basic symbol>	= <non ' '> ' ,
<non ' '>	= <delimiter> <letter> <digit> . ₁₀ ⁺ ₁₀ ⁻ 0 1
<delimiter>	= <standard symbol> <extension symbol> <mark>
<standard symbol>	= <u>begin</u> <u>end</u> () [] () [] <u>before</u> <u>left</u> <u>after</u> <u>right</u> <u>effect</u> <u>real</u> <u>bits</u> <u>string</u> <u>reference</u> <u>array</u> <u>structure</u> <u>procedure</u> <u>precision</u> <u>exact</u> <u>varying</u> <u>nil</u> <u>code</u> <u>by</u> <u>let</u> <u>be</u> <u>represent</u> <u>operate</u> <u>all</u> <u>go to</u> <u>dummy</u> ; : . , _

<extension symbol> == integer | none | comment | silent | 2 array | etc.

<mark> == . | := | ← | if | then | else | for | do | from | step | until |
while | case | of | + | - | × | / | ÷ | ↑ | 7 | ^ | √ | → | ⇒ | < | ≤ | = | ≥ |
> | ≠ | complex | copy | enproc | mode | length | bd | succ |
the | ref | has type | as type | etc.

<code body> is not specified.

1.3 Extensions

- 1.3.1 When E_i is empty or an <expression> for $i = 1, 2$, "real [E_1 : : E_2]" may be replaced by "integer [E_1 : E_2]".
- 1.3.2 "integer [:]" may be replaced by "integer".
- 1.3.3 When J is a <number>, "real J" may be replaced by "J".
- 1.3.4 "precision" may be omitted.
- 1.3.5 When J is a <bits>, "bits J" may be replaced by "J".
- 1.3.6 When J is a <string>, "string J" may be replaced by "J".
- 1.3.7 "exact" may be omitted.
- 1.3.8 "reference nil" may be replaced by "nil".
- 1.3.9 "array (" may be replaced by "(".
- 1.3.10 "structure (" may be replaced by "(".
- 1.3.11 When E_1 and E_2 are <expression>'s, "[E_1 : E_2] array [" may be replaced by "array [E_1 : E_2], "
- 1.3.12 When E_1 and E_2 are <expression>'s, "[E_1 : E_2] array array " may be replaced by "array [E_1 : E_2] array "
- 1.3.13 "][" may be replaced by " , "
- 1.3.14 "array array" may be replaced by "2 array".
"array array array" may be replaced by "3 array". etc.
- 1.3.15 ") effect" may be replaced by ") "
- 1.3.16 ") effect" may be replaced by ") "
- 1.3.17 When V_1, \dots, V_n are <variable>'s and E is an <expression>,
"let V_1 be E ;
let V_2 be E ;
.....
let V_n be E ; "
 may be replaced by
"let V_1, V_2, \dots, V_n be E ; ".
- 1.3.18 Let V be a <variable> or a sequence of <variable>'s separated by commas, and E be an <expression>. If the right-most symbol of E is

effect, real, integer, bits, string, reference,
nil, end, l, l, l, l or'

then "let V be E ; "may be replaced by"EV ;".

1.3.19 Let T_i be a <typifier> for $i=1,2,\dots,n$;

P_i be $\begin{cases} \text{a sequence of <mark>'s for } i=1,2,\dots,n-1 ; \\ \text{empty or a sequence of <mark>'s for } i=0, n. \end{cases}$

"let $P_0(T_1)P_1(T_2)P_2\dots P_{n-1}(T_n)P_n$ represent" may be replaced by

"let $P_0()P_1()P_2\dots P_{n-1}()P_n$ represent".

1.3.20 When G is a sequence of mark 's and parentheses, and E_1,\dots,E_n are <expression>'s,

"let G represent E_1 ;

let G represent E_2 ;

.....

let G represent E_n ;"

may be replaced by

"let G represent E_1, E_2, \dots, E_n ;".

1.3.21 When P_1, \dots, P_n are <mark>'s and Z or Z' is a <left priority> or a <right priority> respectively,

"let P_1 operate ZZ' ;

let P_2 operate ZZ' ;

.....

let P_n operate ZZ' ;"

may be replaced by

"let P_1, P_2, \dots, P_n operate ZZ' ;".

1.3.22 "before left" may be replaced by "before none left".

1.3.23 "after right" may be replaced by "after none right".

1.3.24 Let n be an integer (≥ 0), U_i be a <basic symbol> other than comment and silent for $i=1,2,\dots,n$.

"comment $U_1\dots U_n$ silent" may be inserted between two symbols.